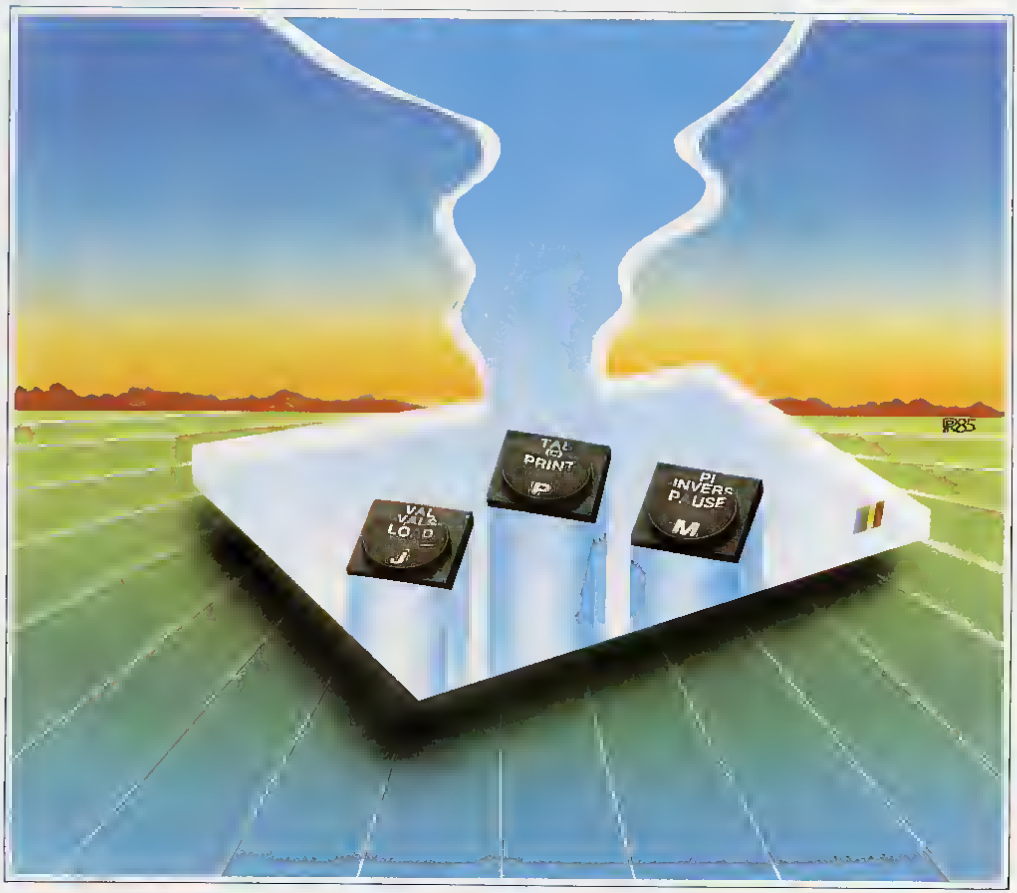
19
150pts.

# RUN

## Enciclopedia Práctica del Spectrum




Nueva Lente/Ingelek

# MINI PROGRAMAS

HEMOS visto hasta ahora que, si nosotros no indicamos lo contrario, el curso normal de ejecución del programa es la interpretación secuencial, línea a una, de todas y cada una de las instrucciones que lo componen, hasta alcanzar la última.

La única forma que conocemos de impedir que esto suceda así, es incluir en alguna parte del programa una sentencia **GO TO**. Como sabemos, **GO TO** permite dirigir la ejecución del programa, tanto hacia adelante como hacia atrás, justo al comienzo del número de línea que indiquemos como parámetro de la sentencia.

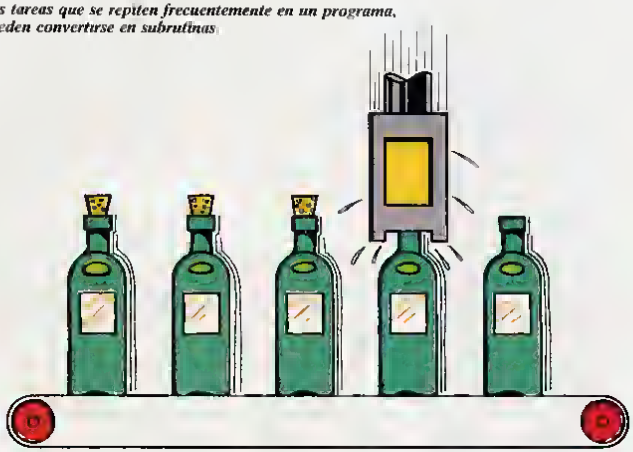
Esta facultad nos permite saltarnos determinado bloque de instrucciones, o ejecutar un conjunto de ellos tantas veces como deseemos (bucle), en base a determinada condición, o de forma imperativa.

En cualquier caso, la sentencia **GO TO** no nos permite desviar la ejecución normal del programa a un bloque concreto de instrucciones, y conseguir que el ordenador «recuerde» desde que punto del programa se le envió, para al concluir la ejecución de este «mini programa», volver al punto de partida y continuar con el programa principal.

*Tanto el* **GO TO** *como el* **GO SUB** *interrumpen la secuencia normal de programa*

Para resolver situaciones de este tipo, el BASIC cuenta con la sentencia **GO SUB** (*GO SUBroutine*) que, como **GO TO**, va seguida del numero de línea a saltar, pero con la condición de recordar desde qué punto exacto del programa se le envió, de forma que al finalizar la ejecución del conjunto de instrucciones deseado, retorne al punto de partida.

*Las tareas que se repiten frecuentemente en un programa, pueden convertirse en subrutinas*



A este conjunto de instrucciones que ha de ejecutarse desde un punto concreto del programa, y con vuelta posterior a su origen, se le denomina SUBRUTINA.

No existe limitación en cuanto al lugar donde colocar las subrutinas de un programa BASIC, pero sí debe especificarse correctamente su principio y final. Es decir, las subrutinas pueden encontrarse al principio, al final, o en la mitad de un programa, e incluso desperdigadas sin orden ni concierto aparente, siempre y cuando sepamos su punto de partida y de final.

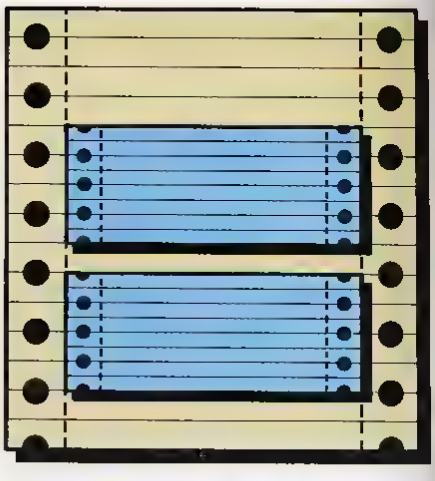
Una subrutina debe comenzar obligatoriamente al principio de una línea de programa, lo mismo que sucede con **GO TO**, puesto que el BASIC no es capaz de bifurcar a una parte concreta de una línea de instrucción. De ser así, la referencia de punto de destino habría de ser más completa, al tener que diferenciar dentro de una misma línea

las diversas instrucciones separadas por dos puntos (:).

Para identificar el final debemos incluir, en la última línea del bloque, una sentencia **RETURN** (en inglés, vuelve). Esta sentencia abandona la subrutina, y regresa exactamente a la siguiente instrucción a aquella en que se ejecutó el **GO SUB** (en inglés, vete a la subrutina) que la envió.

A pesar de que la colocación dentro del programa queda a discreción del programador, hay una regla de oro que debe respetarse, y es la obligación de colocar la subrutina en un lugar en el cual sólo sea posible el acceso a través de las instrucciones **GO SUB** del programa principal.

Debe evitarse a toda costa que el programa pueda entrar en una subrutina por otro medio diferente de **GO SUB**, como podría ser un **GO TO** o una entrada de la propia secuencia normal del programa en la subrutina de forma incontrolada. Caso de producirse este hecho, obtendríamos el mensaje de error **RETURN without GO SUB** (RETURN sin GO SUB).



*Las subrutinas se pueden considerar como «mini-programas» dentro de otro principal*

## LA PILA DE RETORNOS O *STACK*

El **BASIC** dispone de un curioso sistema para gestionar las «idas» y «venidas» a subrutinas, denominado pila de retornos o *stack*. Para entender el funcionamiento del *stack* de forma sencilla, supongamos que nosotros mismos somos el ordenador y estamos ejecutando las instrucciones de un programa, una tras otra.

De improviso nos encontramos con una sentencia **GO SUB**, que nos ordena saltar a determinado número de línea, ¿qué hacer? ¡Está claro!, anotamos en un papel el número de instrucción por el que vamos, lo dejamos encima de la mesa, y pasamos a ejecutar la instrucción que se nos ordena, como si de un **GO TO** convencional se tratara.

Imaginemos ahora, complicando un poco las cosas, que encontramos otra sentencia **GO SUB** dentro de la subrutina. Lo evidente es que, del mismo modo que antes interrumpimos el programa principal para ejecutar una subrutina, debemos actuar ahora en consecuencia.

De esta forma, nos vemos dentro de una estructura anidada, similar a las descritas al hablar de los bucles de programación **FOR-NEXT**, o lo que es lo mismo, inmersos en una subrutina de subrutina.

Para solucionar el problema, recurrimos al procedimiento antes descrito, anotando en otro papel el número de línea en que nos encontramos, y colocándolo sobre la mesa, justo encima del anterior. Hecho esto, podemos efectuar el salto que se nos indica.

Supongamos ahora que tropezamos con un **RETURN**, ¿dónde volver? Pues a la línea indicada en el último papel colocado sobre la mesa, dado que corresponde forzosamente a la última subru-

Cuando un programa entra de forma incontrolada en el área de subrutinas se obtiene el mensaje de error **RETURN without GO SUB** (RETURN sin GO SUB).

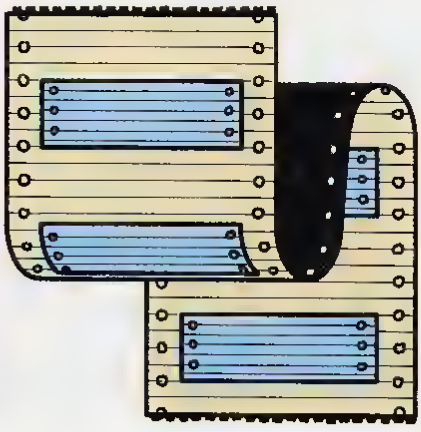
*

El BASIC dispone de un curioso sistema para gestionar las «idas» y «venidas» a subrutinas denominado pila de retornos o *stack*.

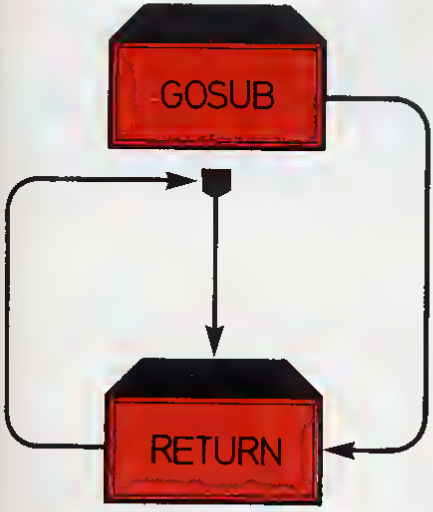
*

El aislamiento de las subrutinas evita que el programa pueda entrar en ellas por otro medio diferente de **GO SUB**, como podría ser un **GO TO** o una entrada de la propia secuencia normal del programa en la subrutina.

*No existe limitación en cuanto al lugar en que se situan las subrutinas dentro de un programa.*



tina en ejecución, por lo cual retiramos el primer papel visible, y continuamos con la ejecución del programa desde el punto ya mencionado
Siguiendo la ejecución del programa, más tarde o más temprano, encontraremos otro **RETURN** y esta vez, tomaremos el último papel que queda sobre la mesa, retornando al programa principal
En base al comportamiento observado, podemos sacar fácilmente la conclusión de que, en el *stack*, el último en entrar es el primero en salir
Este sistema, que de aplicarse en las colas de los cines provocaría auténticas batallas campales, es el idóneo, sin embargo, para el manejo de subrutinas, por permitir su ejecución anidada de forma correcta, y similar a como sucede cuando se anidan bucles del tipo **FOR-NEXT**
Al tipo de almacenamiento de direcciones de retorno en el *stack*, se le denomina *LIFO (Last In First Out)*, en contraposición con el método normal de ejecutar los trabajos que es el *FIFO (First In First Out)*
El manejo de esta pila de retornos no nos debe preocupar en absoluto, dado que es gestionada automáticamente por el Sistema. No obstante, debemos saber que el mantener un gran número de subrutinas «abiertas», es decir sin **RETURN**, puede ocasionarnos, en determinado momento, un problema de desbordamiento del *stack*, produciendo el error **4 Out of memory** (fuera de memoria)

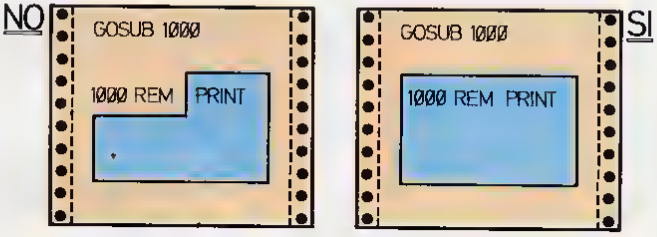


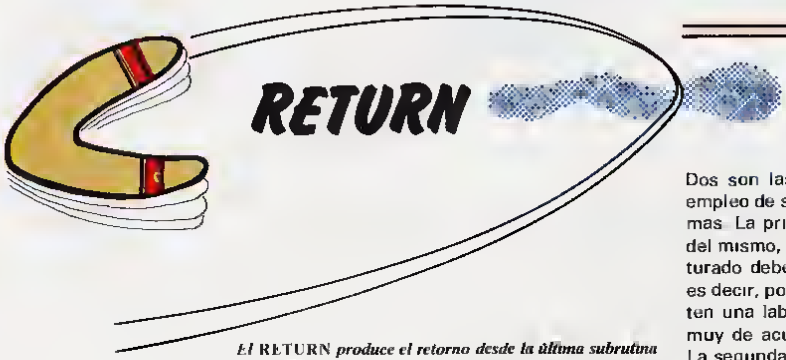

**RETURN** *produce el retorno al punto inmediatamente posterior al de la última llamada*

En la práctica es casi imposible que llegue a darse este problema, puesto que el *stack* del Spectrum es dinámico, y puede crecer desde el final de la memoria RAM hasta su comienzo, tanto como sea necesario
A pesar de ello, se debe vigilar que la salida de las subrutinas se produzca siempre con **RETURN**, y no con una sentencia **GO TO**, puesto que **GO TO** no restaura el *stack* y, aunque el programa funcionará aparentemente de forma normal, no constituye una técnica adecuada de programación
Así pues, para tener una apreciación algo más precisa del funcionamiento concreto del *stack* del Spectrum, debemos imaginar que en vez de situar los papeles con las direcciones de retorno encima de la mesa, utilizamos papeles adhesivos los cuales, una vez escrito el número a recordar, adherimos al techo, en forma de columna, pero no una columna normal, en la cual cuantos más elementos la componen más próxima está del techo, sino todo lo contrario: cuanto más grande es, más se acerca al suelo
Esto se debe a que la información correspondiente a programas, variables, etc... se encuentra al principio de la memoria (digamos que en el «suelo»), si el *stack* se situara a continuación de és-

***El* GO SUB *produce una ruptura «temporal» en la secuencia de programa***

***Las subrutinas siempre deben comenzar al inicio de una línea de instrucción***






*El* RETURN *produce el retorno desde la última subrutina llamada*

## ALGUNOS EJEMPLOS PRACTICOS

Dos son las ventajas principales que ofrece el empleo de subrutinas dentro de nuestros programas. La primera de ellas se refiere a la calidad del mismo, puesto que un programa bien estructurado debe estar codificado de forma modular, es decir, por bloques de instrucciones que ejecuten una labor determinada, y esto se manifiesta muy de acuerdo con la definición de subrutina. La segunda de las ventajas es cuestión de cantidad. Si una misma tarea se repite en diversos puntos de un programa, no sería muy inteligente por nuestra parte escribir las mismas instrucciones todas las veces. El método a seguir en estos casos es programar una subrutina con el bloque de líneas que se repiten, y luego acceder a él siempre que sea necesario mediante **GO SUB**. Como ejemplo de lo comentado vamos a ver las evoluciones de un platillo volador, apoyándonos en el empleo de subrutinas.

```
10 REM SUBRUTINAS
20 LET X=RND*20: LET Y=RND*28
30 LET X$="123333": GOSUB 80
40 PAUSE 10
50 LET X$="       ": GOSUB 80
60 GO TO 20
70 STOP
80 PRINT AT X,Y+1;X$( TO 2); AT X+1,Y
;X$(3 TO ): RETURN
```

*Aunque el comienzo de una subrutina tiene que estar al inicio de un numero de instrucción, su llamada se puede encontrar en otro punto de una línea, realizándose el* RETURN *al punto posterior en la misma línea de instrucción.*

tos, tendría que desplazarse cada vez que hubiera modificaciones en la longitud del programa, el área de variables, etc... Por el contrario, siguiendo el sistema de *stack* «adherido al techo», ambas zonas de memoria pueden funcionar de manera independiente.

Otra posible solución al problema del *stack*, sería establecer una zona determinada de la memoria con destino a él, situada en un lugar fijo, por ejemplo, después de la pantalla y antes del programa. Dado que este área no podría ser utilizada para ningún otro fin, no podríamos hacerla demasiado grande, pues supondría un desperdicio de memoria. De igual modo, si la zona destinada al *stack* es excesivamente pequeña restringiremos el número de subrutinas anidadas.

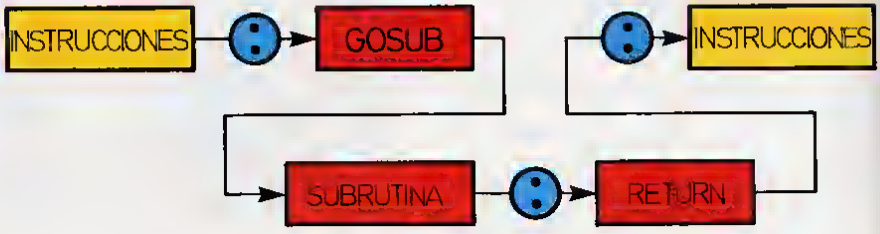
Este último sistema de *stack* es el que adoptan algunos ordenadores, que generalmente pueden tener un máximo de 25 niveles de anidamiento, es decir, 25 llamadas a subrutinas pendientes de retorno. Por el contrario, y a modo indicativo de la eficacia del *stack* dinámico del Spectrum, diremos que en el modelo de 16 K se pueden llegar a alcanzar más de 2.800 anidamientos, y en el de 48 K o PLUS (que en capacidad de memoria son idénticos) ¡más de 13.700!

¡!

Se dice que una estructura de subrutinas es anidada, cuando unas se encuentran completamente dentro de otras, de forma similar al anidamiento descrito para los bucles del tipo **FOR NEXT**.

*

Se denomina «recursividad» a la técnica de programación que se emplea cuando un mismo bloque de programa se llama a sí mismo, cuantas veces sea necesario, hasta resolver el problema concreto que se le encomienda.



En la línea 20 se definen los valores de la abscisa y ordenada para la impresión del platillo. Como podemos ver, en la generación aleatoria se controla que el platillo no pueda caer fuera de los límites de la pantalla.

En la línea 30 se asigna valor a la variable **X$** que contiene la serie de caracteres a imprimir, a través de la subrutina de la línea 80.

La línea 40 tiene el cometido de establecer una pausa, para poder observar las evoluciones del platillo.

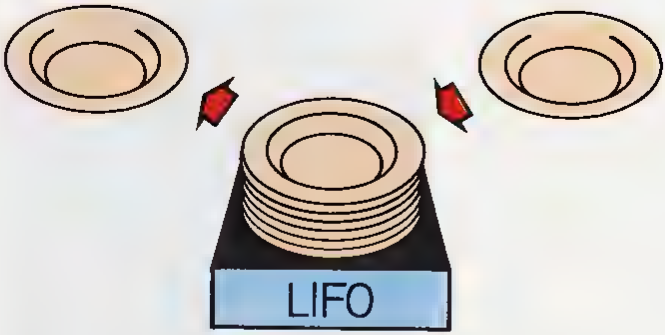
La línea 50 es de contenido similar a la 30, pero esta vez contiene espacios en lugar de los caracteres gráficos que componen el platillo. Con esto logramos, aprovechando una subrutina común, acceder tanto a la escritura como al borrado del móvil.

La línea 60 cierra el ciclo de programa con un **GO TO** imperativo a su comienzo.

La línea 70 tiene una misión fundamental. Como ya dijimos antes, es necesario proteger la zona de subrutinas del programa, para que sólo pueda llegarse hasta ella por medio de las sentencias **GO SUB**.

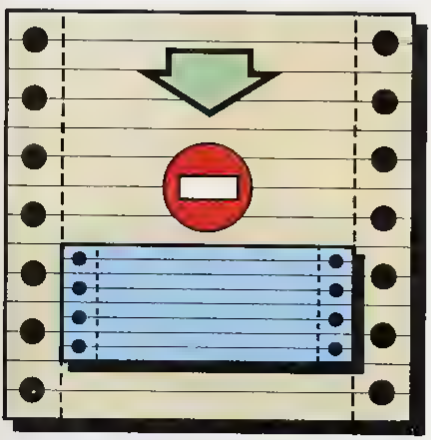
El **STOP** de la línea 70 es, por esta vez, un mero adorno, puesto que el programa nunca pasa por él, evitándose la entrada incontrolada en el **GO TO** de la línea anterior. No obstante, lo incluimos para servir como norma de la separación que debe existir siempre entre las líneas del programa principal y el área de subrutinas.

Por último, en la línea 80 se gestiona la impresión en la pantalla, de acuerdo con las coordenadas **X** e **Y**, del valor de la variable **X$** que, unas veces contiene los gráficos del platillo, cumpliendo la función de escritura, y otras espacios, implicando el borrado de la impresión anterior. El ejemplo propuesto es el más simple de utilización de subrutinas, dado que en el caso más desfavorable, permanece sólo una dirección de retorno en el *stack* del Sistema. Esto es lo mismo que decir que no se produce ningún anidamiento, o sea, una llamada a subrutina desde dentro de otra subrutina.

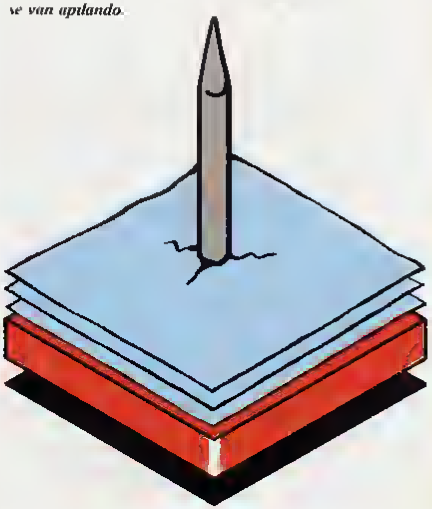



*En el sistema L I F O , el último elemento en entrar es el primero en salir*

*No debemos permitir que el curso del programa se introduzca en el área de subrutinas por un camino distinto al* GO SUB.



*En el stack, las anotaciones con las direcciones de retorno se van apilando.*



¡!

Definir un determinado bloque de programa como subrutina revierte en un importante ahorro de espacio en la memoria pudiendo accederse a él desde cualquier punto del programa.

*

Un programa bien estructurado debe estar codificado de forma modular, es decir por bloques de instrucciones que ejecuten una labor determinada, y esto se manifiesta muy de acuerdo con la definición de subrutina.

*En el sistema F I F O el primer elemento en entrar es el primero en salir*

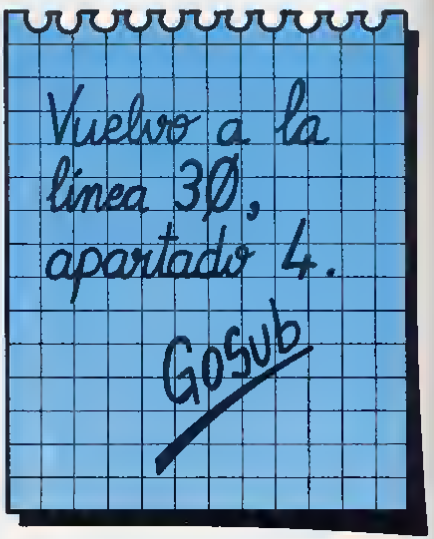
## SOFISTICACION EN LAS SUBRUTINAS

La mayor sofisticación llega cuando una subrutina se codifica de forma recurrente. A esta técnica de programación se le denomina «recursividad», y se produce cuando una misma subrutina se llama a sí misma cuantas veces sea necesario hasta resolver el problema concreto que se le encomienda, de esta forma, al cumplirse el último de los **RETURN** se retorna al programa principal.
Un ejemplo de esta técnica podemos encontrarlo en la resolución del juego de las «Torres de Hanoi». Este juego de habilidad intelectual consta de tres bastidores situados a la izquierda, centro y derecha de la pantalla. En el primero de ellos, se encuentra una torre, de altura variable dependiendo del grado de dificultad de la partida, compuesta por varias piezas en tamaño decreciente, que el jugador debe situar en otro de los bastidores, conservando el mismo orden, y siguiendo unas determinadas normas que a continuación estudiaremos.

Los movimientos se producen peldaño a peldaño, y teniendo cuidado de colocar cada uno de ellos sobre otro de mayor longitud. Lógicamente, como si de una construcción se tratase, nunca puede haber un peldaño mayor sobre otro menor. Siguiendo estas reglas, las piezas se pueden ir pasando a cualquiera de los tres bastidores, hasta conseguir formar la torre inicial en uno de los dos vacíos al comienzo.
La última norma del juego, ilustra claramente el sistema de *stack* L I F O, puesto que sólo podemos mover aquellas piezas que se encuentren sobre cualquiera de las torres, es decir, si queremos desplazar la primera de las piezas, habremos de utilizar primero los movimientos necesarios para apartar las que tenga encima.
Naturalmente, hemos de procurar resolver el problema en el menor número de jugadas posibles, ¡y en eso el Spectrum es un jugador eficacísimo!
Cualquiera que sea el número de piezas de la torre, la cantidad óptima de movimientos seguirá la fórmula **2↑A-1** (dos elevado a **A**, menos uno), donde **A** es la altura de la torre.

```
  10 REM TORRES DE HANNOI - J.M.LOPEZ MA
RTINEZ
  20 INPUT "Altura (2-9):";A: IF A<2 OR
A>9 THEN  GO TO 20
  30 DIM T(3,A): DIM S(A,2): DIM U(3)
  40 LET S$="": FOR I=1 TO A: LET S$=STR
$ I+S$: NEXT I
```

*La información que el Spectrum anota al realizar un* GO SUB, *no es solo el numero de instruccion al que debe regresar, sino tambien el apartado dentro de esta*




**¡!**

Debe respetarse la obligación de colocar la subrutina en un lugar en el cual solo sea posible el acceso a traves de las instrucciones **GO SUB** del programa principal.

*

La sentencia BASIC **GO SUB** va seguida del numero de linea a saltar con la condicion de recordar desde que punto exacto del programa se le envia de forma que al concluir la ejecucion del conjunto de instrucciones deseado retorne al punto de partida.

*

Una subrutina debe comenzar obligatoriamente al principio de una linea de programa lo mismo que sucede con **GO TO**, puesto que el BASIC no es capaz de bifurcar a un segmento de instruccion de los separados por dos puntos ( : ).

```
  50 PRINT TAB 6;"- TORRES DE HANNOI -"
  60 FOR I=1 TO A: LET T(1,I)=I: NEXT I
  70 FOR I=1 TO A: LET S(I,1)=1: LET S(I
,2)=I: NEXT I
  80 LET U(1)=A
  90 GO SUB 130
 100 GO SUB 140
 110 PRINT AT 12,8;"FIN DE PROGRAMA"
 120 STOP
 130 INPUT AT 11,0;"": FOR I=1 TO 3: FOR
 J=1 TO A: PRINT AT 22-J,10*(I-1);("8888
888888")( TO A-T(I,J)+1) AND T(I,J): NEX
T J: NEXT I: RETURN
 140 LET M=VAL S$(LEN S$)
 150 LET T=S(M,1): LET H=S(M,2)
 160 IF H=U(T) THEN  GO TO 190
 170 LET S$=S$+STR$ T(T,H+1)
 180 GO TO 140
 190 LET O=T+(M/2=INT (M/2))-(M/2<>INT (
M/2))
 200 IF O=0 THEN  LET O=3: GO TO 220
 210 IF O=4 THEN  LET O=1
 220 IF NOT U(O) THEN  GO TO 260
 230 IF T(O,U(O))<M THEN  GO TO 260
 240 FOR K=1 TO U(O): IF T(O,K)>M THEN
LET S$=S$+STR$ T(O,K)
 250 NEXT K: GO TO 140
 260 PRINT AT 3,6;"MUEVO DESDE ";T;" HAS
TA ";O: PAUSE 0
 270 LET T(O,U(O)+1)=M
 280 LET T(T,H)=0
 290 LET S(M,1)=O: LET S(M,2)=U(O)+1
 300 LET U(T)=U(T)-1: LET U(O)=U(O)+1
 310 IF LEN S$=1 THEN  GO SUB 130: RETUR
N
 320 LET S$=S$( TO LEN S$-1)
 330 GO SUB 130
 340 GO TO 140
```

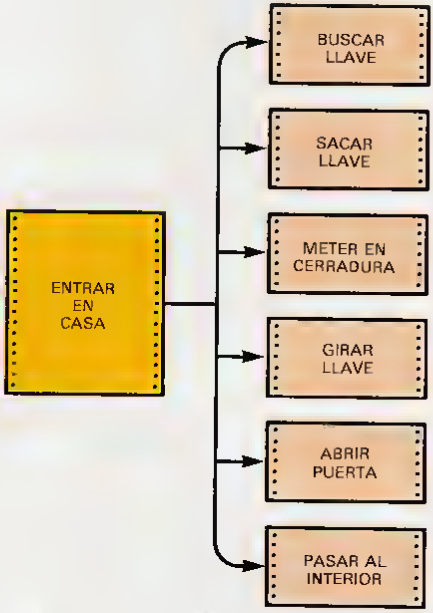
En la línea 20 se solicita la altura de la torre inicial que debe oscilar entre 2 y 9. De no encontrarse la respuesta entre los límites fijados, vuelve a pedirse el dato.

En la línea 30 se dimensionan las tablas **T**, **S** y **U**. Todas ellas son numéricas, y las dos primeras de dos dimensiones, siendo la tercera un vector (unidimensional). En el caso de las dos primeras, el dimensionado depende del valor de la variable **A** (número de peldaños).

La matriz **T(3,A)** almacena el contenido de cada posición dentro del bastidor. **T(2,4)** significa, por ejemplo, el contenido del cuarto peldaño, contando desde abajo hacia arriba, del bastidor central, puesto que los bastidores los hemos numerado 1, 2 y 3, de izquierda a derecha.

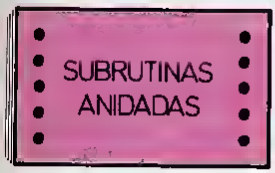
La matriz **S(A,2)** contiene la situación de cada peldaño. El primero de los índices señala la longitud del peldaño, que por tanto, puede oscilar entre 1 y el número de piezas que compongan la torre. Cuando el segundo índice es **1**, se obtiene el número de bastidor en que se encuentra la pieza y cuando el segundo índice es **2**, la altura dentro del bastidor. Así pues, si por ejemplo la pieza de longitud uno, se encuentra en el bastidor central (número 2), y en su base (altura 1), el elemento **S(1,1)**, adoptará el valor 2 (bastidor), y el **S(1,2)**, el valor 1 (altura).

En la línea 40, se efectúa la carga de la variable **S$**, la cual simula el funcionamiento de un *stack*, puesto que almacenamos de entrada en ella los movimientos que es necesario realizar.




*El sistema de la programación estructurada recomienda la división de un problema en subproblemas mas facilmente tratables, éstos pueden ser representados en BASIC mediante subrutinas.*

*Se llaman subrutinas anidadas aquellas que son llamadas desde otras subrutinas*



Al conjunto de instrucciones que ha de ejecutarse desde un punto concreto del programa y con vuelta posterior a su origen, se le denomina SUBRUTINA

*

Se debe vigilar que la salida de las subrutinas se produzca siempre con **RETURN** y no con una sentencia **GO TO** puesto que **GO TO** no restaura el *stack* y aunque el programa funcionara aparentemente de forma normal, no constituye una técnica adecuada de programación.

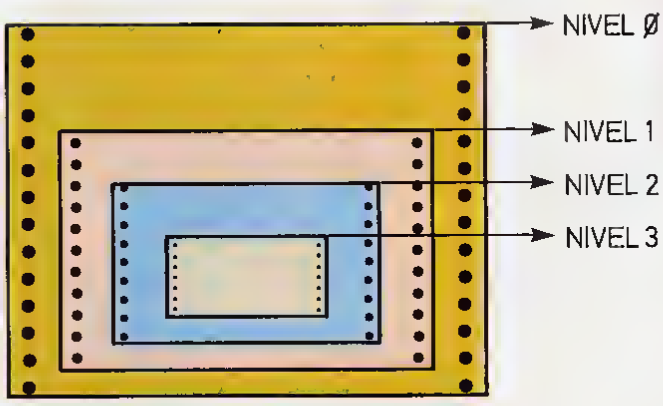
*

Para identificar el final debemos incluir en la última línea del bloque una sentencia **RETURN**. Esta sentencia abandona la subrutina y regresa exactamente a la siguiente instrucción que sigue a la **GO SUB** que le envió.

*El stack dinámico del Spectrum crece desde el final de la memoria hacia al principio*

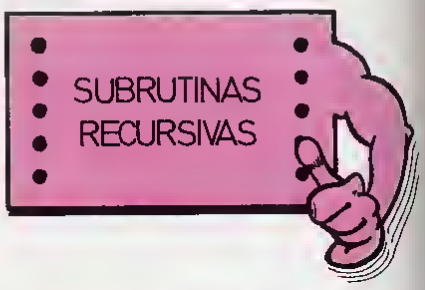
La línea 50 efectúa la presentación de la cabecera de pantalla, y las instrucciones 60 y 70, la inicialización de valores de las tablas **T** y **S**, respectivamente
Para finalizar las inicializaciones, en la línea 80 se asigna la altura actual del primer bastidor (**A**), en el vector **U**. No es necesario definir que las alturas máximas de los otros dos bastidores son cero, dado que al dimensionar una matriz numérica todos sus elementos toman de comienzo el valor cero.
La línea 90 imprime la situación de inicio del juego, definida en la fase de inicializaciones, por medio de un **GO SUB** a la línea 130.
La línea 100 es realmente el núcleo del programa, puesto que contiene un **GO SUB** a la subrutina recursiva 140, que gestiona la totalidad de los movimientos necesarios para solucionar las Torres de Hannoi, en el menor número de movimientos (**2↑A-1**). De hecho, a la vuelta de esta subrutina, se imprime el mensaje de 'FIN DE PROGRAMA', en la línea 110

*El numero de anidamientos de una subrutina, se denomina nivel de anidamiento de la subrutina*




En las líneas 140 a 340 se encuentra la subrutina de movimiento de piezas, hasta alcanzar la solución que llega cuando la variable **S$** tiene un sólo carácter. De no ser así, la subrutina vuelve a ejecutarse por medio del **GO TO 140** de la línea 340.

La variable **M** indica el movimiento a ejecutar, tomado del extremo derecho de la variable **S$**, del mismo modo que **H** indica la altura, extrayéndola de la matríz de posiciones **S**.
En la línea 160 se evalúa si **H** es el peldañó más alto de la torre. De no ser así, se almacena en el «*stack*» **S$** el movimiento de la pieza que impide el desplazamiento deseado, recurriendo a la propia subrutina para calcular un nuevo movimiento.
En las líneas 190 a 210 se evalua el destino **D**, cuando es posible el movimiento, en base a que el peldaño a desplazar sea o no impar. Los pares mueven siempre al bastidor de su derecha y los impares al de su izquierda, controlándose el hecho de que, un desplazamiento a la derecha en el tercer bastidor implica el movimiento al primero, del mismo modo que el desplazamiento a la izquierda, cuando nos encontramos en el primero, implica el salto al tercero.




*La técnica de la recursividad consiste en el acceso retenido a un determinado bloque desde el mismo, hasta la resolución del problema que se le ha encomendado*

De esta forma, el programa sigue filtrando movimientos, almacenando los imposibles por impedimento de otros peldaños en **S$** y efectuando los posibles, describiendo el movimiento a efectuar en la parte superior de la pantalla. Una vez realizada y notificada cada jugada, el programa espera la pulsación de una tecla para continuar con la resolución del problema.

# EL CODIGO MAQUINA

SIN duda alguna, para muchos usuarios del Spectrum, el código máquina es una extraña brujería, la cual les permite llevar a cabo en su aparato tareas que de otro modo serían totalmente imposibles. Esta sensación se traduce en un cierto miedo a la hora de manejarlo, y desde luego, a algunos errores de base que producen a la larga fracasos a la hora de intentar incorporar subrutinas en este lenguaje a nuestros programas. Vamos a continuación a aclarar algunas ideas sobre el tan temido código máquina.

Hemos de comenzar diciendo que, cualquiera que sea el lenguaje de programación que manejemos, éste acabará siendo convertido en código máquina. Para comprender esta afirmación, tenemos que partir de una cierta base sobre el funcionamiento de nuestro ordenador.

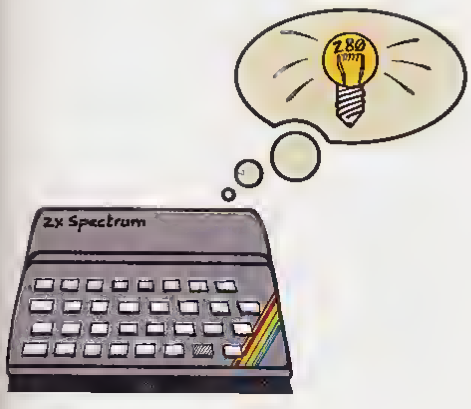
Nuestro aparato inicialmente, cuando está desconectado, no es más que un conjunto inanimado de componentes electrónicos. Ahora bien, al fluir la chispa eléctrica por sus circuitos puede convertirse en un juguete, un profesor, o incluso un amigo ¿a qué es debido este prodigioso proceso de transformación?

El cerebro de nuestro ordenador es el microprocesador Z-80A, un pequeño *chip* capaz de efectuar por sí solo las operaciones que en el albor de la informática realizaba una máquina que ocupaba toda una habitación. Al igual que los humanos entendemos un determinado lenguaje en el que se nos pueden dar órdenes o explicaciones, el microprocesador habla un lenguaje electrónico en el cual se deben codificar las instrucciones para que él llegue a entenderlas.

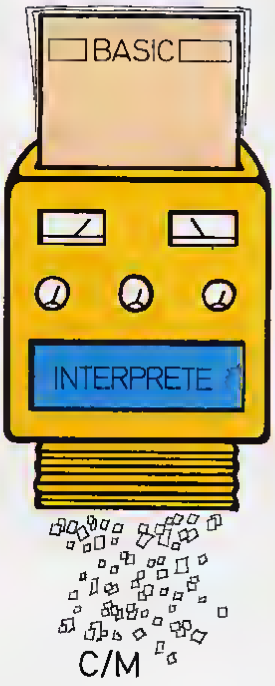
Como habremos supuesto, este lenguaje está formado por impulsos eléctricos, que nosotros representamos por unos y ceros (bits), e intentar comprenderlo nos es relativamente complicado, aunque no tanto como lo sería para el ordenador comprender el nuestro, ello se debe a que el lenguaje del Z-80 es incomparablemente más limitado que el humano, y carece de imprecisiones: este lenguaje se denomina LENGUAJE MAQUINA o CODIGO MAQUINA.

*Debemos desterrar la idea de que el codigo maquina es algo que funciona por arte de magia*

*El cerebro de nuestro ordenador es el microprocesador Z-80*



## FIRMWARE




*Al fin y al cabo, programemos en el lenguaje que programemos, todo se acabará convirtiendo al código máquina*

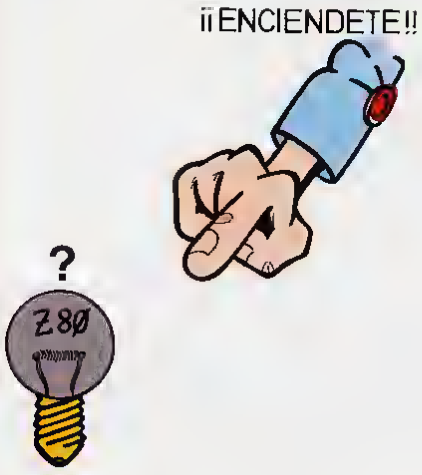
Acabamos de decir que el ordenador es incapaz de comprender nada que no sea el lenguaje máquina, sin embargo, nosotros nos comunicamos en BASIC con nuestro Spectrum. La explicación a este hecho es bien clara; ni los humanos estábamos dispuestos a aprender el lioso código máquina, ni el microprocesador era capaz de comprender directamente el lenguaje humano, de modo que adoptamos una solución intermedia: contratar los servicios de un traductor.

Cuando nuestro aparato se conecta a la alimentación, lo primero que hace el microprocesador es acceder a una memoria denominada ROM, en la cual se encuentran almacenadas las instrucciones referentes a lo que debe hacer a partir de ese momento. Naturalmente, estas indicaciones se encuentran en código máquina, para que puedan ser comprendidas directamente por el ordenador.

Todas las tareas que se efectúan al encender se encuadran en un proceso que se denomina INICIALIZACION, y van desde el borrado de la memoria RAM (del usuario), hasta la impresión del mensaje de presentación (© *1982 Sinclair Research Ltd*). Ahora bien, en la ROM no sólo se encuentra esta información, sino también otras muy interesantes, el MONITOR, que nos permite la comunicación con la CPU a través del teclado, el INTERPRETE del BASIC, la forma que tienen los caracteres (GENERADOR DE CARACTERES), rutinas matemáticas (CALCULADOR) que le enseñan al microprocesador como multiplicar, hacer raíces cuadradas, etc...

Al conjunto de informaciones que el fabricante ha depositado en la ROM, se le denomina FIRMWARE y es algo así como los conocimientos básicos del ordenador, que nosotros podemos combinar mediante el SOFTWARE en RAM, para conseguir realizar tarás más complicadas (juegos, programas de contabilidad, etc...) En definitiva, el FIRMWARE no es más que un complicado programa en código máquina (ocupa 16 K de memoria), que contiene los conocimientos de partida del ordenador.

*El microprocesador no comprende las órdenes que le damos directamente en nuestro lenguaje, ni tan siquiera en BASIC.*




**¡!**

A los circuitos integrados se les denomina familiarmente *chips*

*

El lenguaje que maneja directamente un microprocesador se conoce como CODIGO MAQUINA o LENGUAJE MAQUINA. En muchas ocasiones encontraremos su abreviatura C/M o la abreviatura inglesa M/C *(machine code)*

*

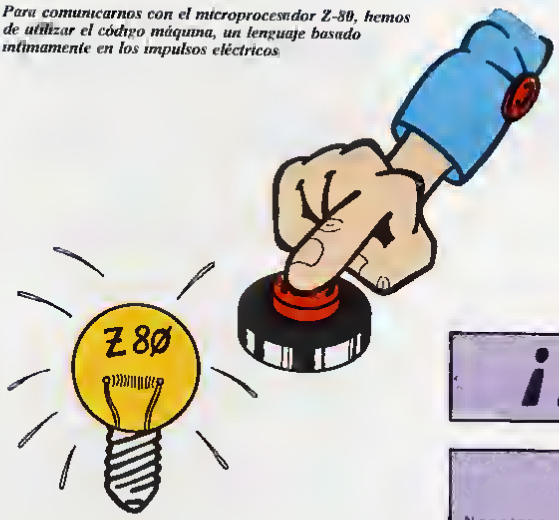
El software que el fabricante incorpora en la memoria ROM de nuestro aparato se denomina *firmware*

*

Las instrucciones del codigo máquina no se expresan con letras y palabras, sino mediante números

## UN LENGUAJE DE NUMEROS

*Para comunicarnos con el microprocesador Z-80, hemos de utilizar el código máquina, un lenguaje basado intimamente en los impulsos eléctricos*

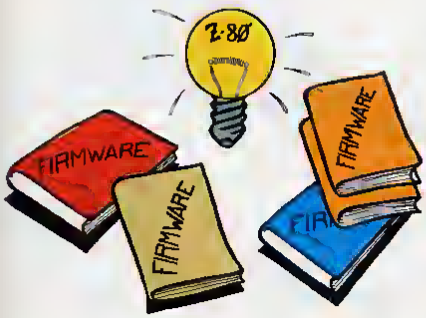



Así pues, dado lo que acabamos de ver, estamos en condiciones de comprender por que cualquier taréa que llevemos a cabo en el ordenador, se encuentra codificada en lenguaje máquina. Así por ejemplo, cuando pulsamos una tecla, una parte del firmware se encarga de notificarle al ordenador cual de ellas hemos pulsado, de forma parecida, al introducir un programa BASIC, estamos ejecutando la rutina en código máquina, dentro de la ROM, denominada EDITOR, y cuando ejecutamos el programa BASIC que hemos introducido, ponemos en funcionamiento el INTERPRETE para que traduzca las instrucciones a código máquina

El lenguaje máquina no está compuesto por palabras como el BASIC, sino por números. Si le decimos al Z-80 que ejecute la instrucción 60, él entenderá que debe incrementar en uno el contenido de la variable A, es decir, algo así como si en BASIC escribiéramos LET A=A+1; si por ejemplo tuviera que ejecutar un código 201, lo entendería como un retorno de subrutina, es decir, exactamente igual que un **RETURN** en el BASIC. Como podemos ver, el código máquina no es cosa de brujería, sino simplemente un lenguaje más, con algunas similitudes con el BASIC, aunque mucho más restringido. Cuando lo estudiemos más en profundidad, tomaremos una conciencia completa de su funcionamiento, y veremos que la variable A de la cual hemos hablado en el ejemplo de unas líneas más arriba (ejecución de un código 60), no es la variable BASIC A que nosotros hemos podido definir, ¡ni tan siquiera se le parece! Las diferencias son muchas, por ejemplo, esta «variable», que más propiamente se denomina REGISTRO, sólo puede contener un número natural entre 0 y 255.

Está claro que las «variables» del codigo máquina, son bastante menos útiles que las del BASIC sin embargo programando en el lenguaje del microprocesador no disponemos nada más que de este tipo de «variables», y lo que es más, sólo disponemos de ocho de estos registros, todos ellos numéricos, y de ninguno de manejo con cadenas. Del mismo modo que utilizamos las instrucciones básicas del BASIC para conseguir programas más complejos, se pueden combinar las instrucciones básicas del código máquina para confeccionar programas más complejos, aunque lógicamente, al partir de una base más pobre, llegar a los mismos resultados necesita en este último caso mayores esfuerzos

*Los conocimientos básicos del ordenador se encuentran almacenados en el firmware*




## PASO DEL BASIC AL CODIGO MAQUINA

De lo dicho hasta ahora se desprende que siempre, estemos o no ejecutando un programa en có-

**¡!**

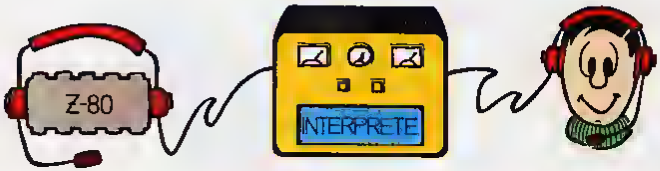
Nosotros podemos crear nuestras propias subrutinas en codigo máquina, depositandolas en cualquier zona de la memoria RAM

*

Para la ejecución de una subrutina en codigo máquina se utiliza la funcion BASIC USR seguida de la primera direccion de memoria a interpretar

*

Al ejecutar una subrutina en código máquina puesto que no nos encontramos bajo el control del firmware de nada servira que intentemos detener el programa mediante **BREAK (CAPS SHIFT + SPACE)**




*Gracias a la zona del firmware denominada intérprete, podemos comunicarnos con el ordenador en lenguaje BASIC.*

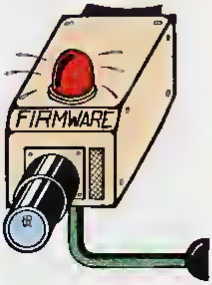
digo máquina, nos encontramos bajo el control del lenguaje máquina, más concretamente, bajo el control del FIRMWARE. No obstante, existe una manera de evitarlo, la función BASIC **USR**. Esta nos permite ejecutar una subrutina propia en código máquina, situada por tanto en la RAM, la cual terminará habitualmente con un código 201, que como sabemos es un retorno al punto desde el cual se llamó la subrutina, en nuestro caso concreto, este punto de llamada fue el BASIC, lo que significará recuperar el control completo por parte del *firmware*.

El utilizar la función **USR** tiene sus ventajas y sus inconvenientes: por una parte, al ser el código máquina un lenguaje mucho más simple, el tiempo de ejecución es considerablemente menor, es decir, un programa confeccionado en código máquina siempre funciona más rápido que uno realizado en BASIC y que cumpla un cometido idéntico.

Por otro lado, cada vez que algo no funciona bien, estando bajo el control del *firmware*, recibimos un mensaje de error, bastante documentado. Sin embargo, si no tenemos mucho cuidado, y contemplamos todas las posibilidades de error (cosa por otra parte no del todo fácil), puede ser que ante circunstancias extrañas el código máquina nunca regrese al BASIC, con lo cual nuestra única solución será apagar y encender el ordenador, para recuperar nuevamente el control.

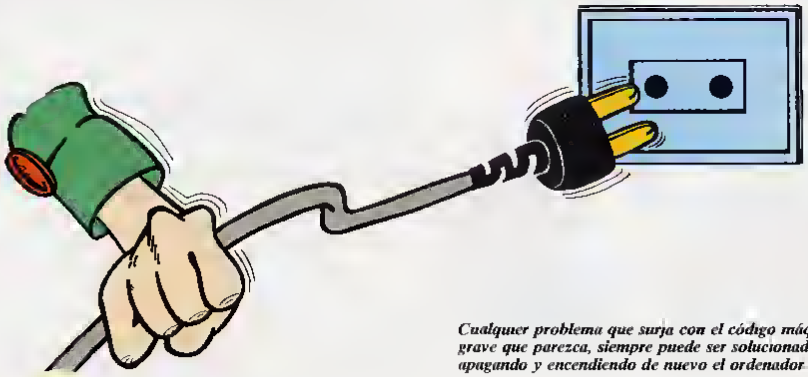
No obstante, en estos casos perdemos la información que hubiera en la memoria, y por tanto, es conveniente que en prevención de posibles problemas, antes de ejecutar un programa en código máquina, conservemos una copia en cinta del mismo. En todo caso, las posibles «catástrofes» que produzca el código máquina, al igual que las del BASIC, nunca llegarán más allá de tener que apagar y encender el ordenador, jamás le producirán un deterioro permanente, así pues ¡sin miedo! ¡experimentemos!

## CONTINUARA...




*El firmware vigila constantemente nuestras operaciones en BASIC, procurando facilitarnos el trabajo.*

Pronto adquiriremos ya las nociones suficientes como para no tener ningún miedo al código máquina, e incluso haremos algún que otro miniprograma. Hasta entonces conviene que intentemos asimilar bien los conocimientos hasta ahora expuestos sobre el tema, puesto que serán la llave para abrir la puerta de este nuevo lenguaje. Por último, adelantaremos que se van a tocar conceptos de gran interes, como la REUBICABILIDAD, o el significado del ENSAMBLADOR, desentrañando ciertos misterios sobre el código máquina que aún permanecen sin desvelar.



*Cualquier problema que surja con el código máquina, por grave que parezca, siempre puede ser solucionado apagando y encendiendo de nuevo el ordenador.*

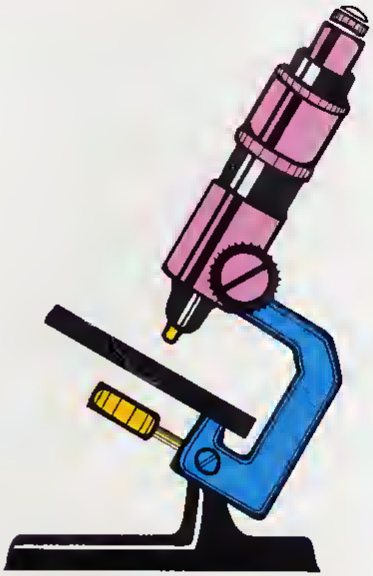
# CARACTERES GIGANTES

EN multitud de ocasiones, nuestros propios programas se ven necesitados de alguna subrutina capaz de potenciar la estética de los mismos, así como acentuar ostensiblemente la legibilidad de los mensajes o informaciones que aparecen en ellos.

A pesar de que ya hemos hecho uso de esta subrutina con anterioridad, es muy interesante disponer de un programa que nos explique la forma en que podemos incluirla en nuestros programas, sus espectaculares efectos habrán causado sensación entre los jugadores de LA BOMBA, y algunos otros programas. Todo ello no ha sido más que un adelanto de lo que se puede conseguir con esta subrutina de la firma británica PSION Computers, que se incluye a modo gratuito con la casete de demostración de los ZX Spectrum.

Las posibilidades de la subrutina son muy grandes, puesto que no es sólo posible generar literales de las dimensiones que hemos visto en los mencionados programas, sino seleccionar a nuestro gusto su anchura, su altura y su punto de posicionamiento en la pantalla, teniendo este sistema de situacion precisión de *pixel*.

Veamos a continuación la forma de experimentar con el programa de demostración. Ello nos servirá para pasar a continuacón a adaptar la subrutina a nuestros propios programas.



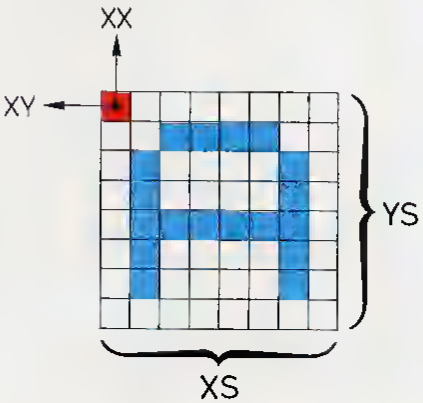
*La subrutina de caracteres gigantes, permite aumentar a nuestro gusto las dimensiones de los caracteres estándar.*

## TRABAJANDO CON EL PROGRAMA

En primer lugar, deberemos introducir correctamente todas las líneas BASIC que componen el programa, y a continuación, seguir el sistema explicado en el programa LA BOMBA para la adopción de la subrutina en código máquina. Es recomendable antes de la primera ejecución de cualquier programa que incluya código maquina, conservar una copia grabada del mismo, puesto que en caso de haber cometido algún error de introducción, los resultados pueden ser catastróficos.

Una vez ejecutado el programa, comenzará en la

*Los factores que intervienen en la representación de caracteres son XX, YY, XS e YS*




## ¡!

Tanto los numeros de instruccion que se utilizan para el soporte BASIC del código maquina, como las variables utilizadas para indicar la información referente al rótulo, pueden ser alteradas segun las necesidades del programa concreto a que se aplique.

*

Indicando como factores de proporcionalidad vertical y horizontal (YS y XS) el valor uno, es posible emular los caracteres estandar del Spectrum, con la ventaja adicional de poder situarlos en cualquier punto de la pantalla.

¡!

Aunque nuestro modelo sea de 48 K el programa de demostración esta preparado para funcionar solo con la subrutina para 16 K

*

La dirección de carga así como la de ejecución ha de ser exactamente la especificada en el programa

pantalla una secuencia de entrada de informaciones, mediante sentencias **INPUT**. Los datos a teclear son los siguientes:

1. ALTURA DEL CARACTER: el número introducido indicará las posiciones de carácter que tendrá el literal deseado en vertical. Dicho número ha de estar comprendido por tanto entre 1 y 21. Dado que la altura de un carácter estándar es la unidad, podemos decir que este factor, equivale al numero de veces por el cual se multiplica la altura del mensaje (2 el doble, 3 el triple, etc.).
2. ANCHURA DEL CARACTER: este dato es muy similar al anterior, aunque concerniente a las dimensiones en horizontal, así pues deberá estar comprendido entre 0 y 31. Lógicamente, este factor es equivalente al número por el cual se multiplica la anchura estándar del mensaje.
3. ALTURA DEL LITERAL: determina la posición (en *pixels*), en la cual se escribe el punto superior izquierdo del primer carácter a representar, por tanto, los valores han de estar comprendidos entre 0 y 192. Tengamos en cuenta, que al contrario que las coordenadas de **PLOT**, este parámetro toma como punto cero la línea superior de la pantalla.
4. LITERAL CENTRADO: ? como su nombre indica, centrará automáticamente el rótulo en horizontal. En el caso de que no aceptáramos esta opción, el ordenador nos pedirá el *pixel* que define el margen izquierdo del mensaje. Dicho pixel es el perteneciente a la esquina superior izquierda del primer carácter del rótulo a mostrar, y a partir del cual se imprime en pantalla todo el literal.
5. COLOR DE LA TINTA: imprimirá el rótulo con el color de tinta deseado.
6. COLOR DEL PAPEL: representará el mensaje con el fondo indicado.
7. BRILLO: ? imprimirá el mensaje con o sin brillo, dependiendo de la opción escogida.
8. FLASH: ? presentará el mensaje estable o con intermitencia, según nuestra indicacion.
9. INTRODUCE LITERAL: por último, teclearemos el rótulo que queramos imprimir en la pantalla.

En lo referente a todos estos datos, hemos de destacar que cada uno de los literales suceptibles de ampliación, ha de ser cuidadosamente estudiado, con el fin de que su aparición en la pantalla sea correcta. En este sentido, es muy importante que el literal no salga del margen inferior de la pantalla, puesto que ello produciría la prácticamente segura «caída del Sistema». En horizontal no tenemos por qué tener esa precaución, dado que simplemente se sobreescribirán los caracteres del mensaje en la misma línea.
En cuanto a los **INPUT** de la toma de datos en los cuales se nos propone alguna opción, deberemos introducir SI o NO en respuesta a la pregunta que se nos realiza. Una vez finalizada la secuencia de entrada de datos, obtendremos un resumen de la misma, destinada a la comprobación de los valores tecleados. Si éstos no fuesen los correctos, se lo haríamos saber al programa pulsando **N**, con lo cual retornaríamos a la secuencia de introducción.

## ADAPTACION A NUESTROS PROGRAMAS

Para el acceso de la subrutina de utilidad en CODIGO MAQUINA es necesario un pequeño soporte BASIC, cuyo listado aparece a continuación:

```
10 LET I=23306: POKE I,XX: POKE I+1,YY
: POKE I+2,XS: POKE I+3,YS
20 POKE I+4,8: LET I=LEN P$: FOR K=1 TO
I: POKE 23310+K, CODE P$(K): NEXT K
30 POKE 23311+I,255: LET I=USR 32256
```

La numeración de dichas instrucciones puede ser alterada por nosotros, con el fin de incluirla en la zona oportuna de nuestros propios programas. Dentro de la citada subrutina BASIC, las variables utilizadas tendrán los siguientes valores:

YS= FACTOR MULTIPLICADOR DE ALTURA
XS= FACTOR MULTIPLICADOR DE ANCHURA
YY= PIXEL DE COMIENZO EN ALTURA
XX= PIXEL DE COMIENZO DE ESCRITURA EN HORIZONTAL
P$= ROTULO A VISUALIZAR

Los poseedores del modelo de 16 K, no tendremos más que incluir a continuación de todos los programas que utilicen la subrutina, el código máquina, tal como se explica en el programa la BOMBA.
Este sistema funcionará también en el modelo de 48 K, si bien es cierto que al situarse la subrutina a partir de la dirección 32256, limita la capacidad de nuestra memoria. Si queremos adaptar el código máquina a las 48 K de que disponemos, deberemos realizar las siguientes operaciones:

— Utilizar un programa cargador diferente al facilitado para el programa la Bomba.

```
10 REM CARGADOR DE CODIGO MAQUINA
20 CLEAR 64868
30 FOR I=64869 TO 65145
40 INPUT A
```

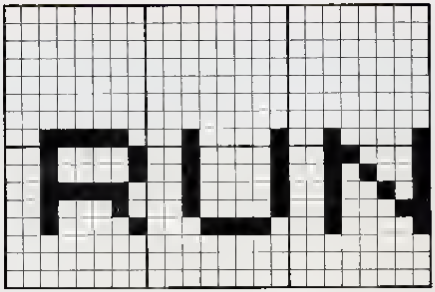
```
50 POKE I,A
60 NEXT I
```

— Efectuar estos POKEs

| | |
|---|---|
| POKE 64955,104 | 64956,253 |
| POKE 64975,133 | 64976,253 |
| POKE 64996,9 | 64997 254 |
| POKE 65023,149 | 65024,253 |
| POKE 65120,1 | 65121,254 |

Estos adaptarán nuestra subrutina casi al tope de memoria 64860 Por tanto la dirección de llamada (**USR**) que aparece al final de la subrutina BASIC debera ser también 64869 Del mismo modo la proteccion mediante **CLEAR**, como se puede observar al comienzo del nuevo cargador, será 64868

De entre los muchos efectos curiosos que se pueden obtener con la subrutina, destacaremos dos por una parte indicando XS=1 e YS=1 se simulan las dimensiones normales de los caracteres




*Dando valor uno a los factores* **X$** *e* **Y$**, *podemos escribir caracteres estándar «a caballo» entre varias posiciones de pantalla*

del Spectrum, con la ventaja de poder seleccionar por *pixels* el comienzo de escritura de los mismos, tanto en vertical como horizontal, esto nos permitirá escribir mensajes «a caballo» sobre varias posiciones de carácter, o incluso escribir en lugares difícilmente accesibles por el BASIC, como son las líneas reservadas al Sistema, y aún más, la línea de separación entre ambas zonas de la pantalla

Por último el **POKE I+4,8** que señalamos al comienzo de la línea 20 en el soporte BASIC sugerido, indica la separación en puntos entre los caracteres consecutivos de un mensaje a tamaño normal. Variar este valor, puede suponer conseguir mensajes montados, o con mayor separación entre letras, así como escritura inversa (de derecha a izquierda), en caso de utilizar valores negativos.

¡!

Es muy importante elegir cuidadosamente las dimensiones y punto de comienzo del rotulo, para que este no exceda los límites de pantalla por su zona inferior

*

Recordemos cargar la subrutina en código maquina antes de ejecutar el programa de demostración, de no hacerlo así sin duda tendremos que volver a cargar el programa

```
  10 REM **********************
  20 REM * J.M.MAYORAL SERRANO *
  30 REM **********************
  40 REM * CARACTERES GIGANTES *
  50 REM **********************
  60 REM  PROGRAMA PRINCIPAL
  70 LET SW=0: DIM D(4): GO SUB 3440
  80 DATA "ANCHURA DEL CARACTER . . .= ","ALTURA DEL CA
RACTER . . . = ","ALTURA DEL LITERAL .....= ","LITERAL
CENTRADO . . . . . = "
  90 DATA "COLOR DE TINTA . . . . = ","COLOR DEL PAPEL
. . . = ","BRILLO . . . . . = ","FLASH . . . . .
. . . . . = "
 100 RESTORE 80
 110 FOR N=1 TO 3
 120 READ A$
 130 PRINT
 140 PRINT TAB 1;A$;D(N)
 150 NEXT N
 160 READ A$
 170 PRINT
 180 PRINT TAB 1;A$;" PAPER 2, INK 6; FLASH 1;B$
 190 PRINT
 200 IF B$="NO" THEN  PRINT TAB 1;"SEPARACION MARGEN IZQD
O . . .= ";XX: GO TO 220
 210 PRINT
 220 GO SUB 3790
 230 PRINT , PRINT
 240 PRINT TAB 6; FLASH 1;" TODO CORRECTO ??"
 250 PRINT TAB 12;"(S/N)"
 260 IF INKEY$="N" THEN  CLS : GO TO 70
 270 IF INKEY$="S" THEN  GO TO 290
 280 GO TO 260
 290 CLS : PAPER P: INK T: FLASH FLASH: BRIGHT BRILLO
 300 GO SUB GS
 310 IF INKEY$="" THEN  GO TO 310
 320 BORDER 0
 330 PAPER 0
 340 INK 9
 350 BRIGHT 0
 360 FLASH 0
 370 RUN
3000 REM SBR. CARACT. GIGANTES
3010 LET xx=(256-R*xs*LEN p$)/2
3020 LET i=23296
3030 POKE i,xx: POKE i+1,yy: POKE i+2,xx: POKE i+3,yy
: POKE i+4,R
3040 LET i=i+4: LET w=LEN p$
3050 FOR q=1 TO w
3060 POKE i+q,CODE p$(q)
3070 NEXT q
3080 POKE i+w+1,255
3090 LET w=USR 32256
3100 RETURN
3110 REM  AUTO-EJEC. OBLIGATORIA
3120 BORDER 0: PAPER 0: CLS
3130 CLEAR 32255
3140 LOAD *"M";1;"LIT"CODE : POKE 23658,8
3150 REM  PRESENTACION
3160 LET P$="CARACTERES"
3170 LET XS=3
3180 LET YS=2
3190 LET YY=0
3200 BRIGHT 1: INK 6: PAPER 2
3210 GO SUB 3000
3220 LET YY=16
3230 LET YS=17
3240 LET XS=4
3250 PAPER 0
3260 LET P$="GIGANTES": INK 4
3270 GO SUB 3000
3280 FOR N=0 TO 200
3290 BORDER 1: BORDER 5: BORDER 6
3300 NEXT N
3310 LET P$="ZX Spectrum $"
3320 LET YY=140
3330 LET XS=2
3340 LET YS=1
3350 PAPER 1: INK 7: FLASH 1
3360 GO SUB 3000
3370 BORDER 0
3380 FOR N=0 TO 50
3390 BEEP .01,10: BEEP .02,20
3400 BEEP .03,30: BEEP .04,40
3410 NEXT N
3420 FLASH 0: PAPER 0: BRIGHT 0
3430 RUN
3440 REM  INTRODUCCION DATOS
3450 INPUT  PAPER 6; INK 9;"ALTURA CARACTER = ";YS: L
ET D(2)=YS
3460 INPUT  PAPER 6; INK 9;"ANCHURA CARACTER = ";XS:
LET D(2)=XS
3470 INPUT  PAPER 2: INK 9;"ALTURA LITERAL = ";YY: LE
T D(3)=YY
3480 INPUT  PAPER 2: FLASH 1; INK 9;"LITERAL CENTRADO
 (S/N)  ";A$: LINE S$
3490 IF CODE S$=83 OR CODE S$=78 THEN  GO TO 3510
3500 BEEP .2,-10: GO TO 3480
3510 IF S$="N" THEN  INPUT "SEPARACION MARGEN IZQDO-"
;XX: LET GS=3020: LET B$="NO": GO TO 3530
3520 LET B$="SI": LET GS=3000
3530 INPUT  PAPER 1; INK 9;"COLOR TINTA = ";T: GO SUB
 3670
3540 IF T>7 OR T<0 THEN  GO TO 3530
3550 INPUT  PAPER 1; INK 9;"COLOR PAPEL = ";P: LET SW
=1: GO SUB 3670
3560 IF P>7 OR P<0 THEN  GO TO 3550
3570 INPUT  INK 6;" BRILLO ??    (S/N)    ";C$
3580 IF C$="S" THEN  LET BRILLO=1: LET C$="SI": GO TO
 3610
3590 IF C$="N" THEN  LET BRILLO=0: LET C$="NO": GO TO
 3610
3600 GO TO 3570
3610 INPUT "FLASH ??     (S/N)    ";F$
3620 IF F$="S" THEN  LET F$="SI": LET FLASH=1: GO TO
3650
3630 IF F$="N" THEN  LET F$="NO": LET FLASH=0: GO TO
3650
3640 GO TO 3610
3650 INPUT  FLASH 1; PAPER 2; INK 6;"   INTRODUCE  LI
TERAL  ";" ";rP$
3660 RETURN
3670 REM
3680 REM
3690 DATA "NEGRO","AZUL","ROJO","MAGENTA","VERDE","CI
AN","AMARILLO","BLANCO"
3700 RESTORE 3690
3710 FOR N=0 TO 7
3720 READ M$
3730 IF SW=1 THEN  IF N=P THEN  LET B$=M$
3740 IF N=T THEN  LET I$=M$
3750 NEXT N
3760 RETURN
3770 REM
3780 REM
3790 RESTORE 90
3800 FOR N=1 TO 4
3810 READ A$
3820 PRINT
3830 PRINT TAB 1;A$
3840 NEXT N
3850 LET SW=0
3860 PRINT AT 11,24;I$
3870 PRINT AT 13,24;B$
3880 PRINT AT 15,27;C$
3890 PRINT AT 17,27;F$
3900 RETURN
```